AMENDMENTS TO THE SPECIFICATION

Please replace paragraph 67 on page 12 lines 3-15, with the following re-written paragraph:

The instrumentation tool 38 inserts instrumentation code into the instrumented class to load an appropriate plug-in instrument object when the instrumented class is initialized. To ensure an instrument object that implements the execCallback interface 36 is loaded before any methods in the instrumented class are called, the instrumentation tool 38 modifies the class initialization method <clinit> of the instrumented class to load the instrumentation object at class initialization time. If the instrumented class does not include a <clinit> method, instrumentation tool 38 inserts one that includes a return. The <clinit> method calls a static method "getHook" (or a method that calls getHook), which either creates the plug-in instrument object or uses a factory object to create the plug-in instrument object. ~~[What is the difference between the getHook method and the BIPinstallHook method???]~~ If the instrumented class does not include a <clinit> method, and the instrumented class implements the Serializable interface, another mechanism is used to load the plug-in instrument object, as described below.

Please replace paragraph 87 on page 17 lines 1-3, with the following re-written paragraph:

At 614, if the class file/archive contains more classes, control passes to 620. Otherwise, control passes to ~~offer page~~ off-page connector "C" 622 in FIG. 6C. At 620, consideration is advanced to the next class, and control passes to 606.

Please replace paragraph 144 on page 29 lines 15-33, with the following re-written paragraph:

By way of example, with reference to FIG. 15, a method of a wrapper class C' that has been instrumented according to the teachings of the invention to include hooks for invoking ARM calls will be registered with an ARM agent 48 that implements the ARM protocol functionality. The registration of the instrumented method can be accomplished by utilizing a set of attributes, such as those described above, that would uniquely identify the instrumented method to the ARM agent 48. The instrumentation code, during execution of the instrumented method, will call the ExecCallback interface 36. In response to such a call, the ExecCallback interface 36 will communicate with the ARM agent 48 to generate an ARM transaction object. For example, immediately prior to starting an instrumented

method or function in class C', the ExecCallBack interface can communicate with the ARM agent 48 to invoke an ARM start) method that allows the ARM transaction object to save a start time marker (timestamp). Further, immediately after the transaction ends, an ARM stop ( ) method is invoked to save a stop time marker (timestamp). The start and stop ~~step~~ time markers can then be utilized to determine the response time of the instrumented method or function. More particularly, upon invocation of the start ( ) method, the ARM agent can initiate a transaction record corresponding to the transaction initiating the call, and upon invocation of the stop ( ) method, the ARM agent can complete the transaction record. As discussed in more detail below, the transaction record can be transmitted to a measurement server for analysis, presentation to a user, and/or storage in a database.

Please replace paragraph 190 on page 42, line 20 to page 43, line 3, with the following re-written paragraph:

With reference to FIG. 26, in this exemplary embodiment, 10 initial bytes of the code associated with the CoCreateInstance function are replaced with a jump (jmp) instruction to a function supplied by the OvCom.dll, herein, referred to OVTACoCreateInstance, schematically depicted in FIG. 27. The 10 bytes of the CoCreateInstance function that have been replaced can include all and/or part of each of a plurality of instructions. Hence, prior to overwriting the initial 10 bytes by the jump instruction, those instructions of the CoCreateInstance function that are wholly or partially included in these 10 bytes, are copied to a data area 2138. Further, a jump instruction is included in the data area 2138, after the copied instructions, that refers to an instruction in the CoCreateInstance function immediately following the last copied instruction. For example, in this exemplary illustration, the XXX, YYY, and ZZZ instructions include 12 bytes. Hence, prior to inserting the jump instruction to OVTACoCreateInstance function, these instructions are copied to the data area 2138. Further, ~~the a jmp~~ a jump instruction is included after the copied instructions in the data area 2138 to refer back to the CoCreateInstance function, or more specifically, to the instruction AAA following the ZZZ instruction ~~instruction~~ in the CoCreateInstance function.